LLNL-TR-654332

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Estimating the Impact of Advanced Architectures on ASC Multiphysics Codes in the Absence of an Exascale Initiative

C. H. Still, C. J. Clouse, M. G. McCoy, J. R. Neely, B. S. Pudliner, J. A. Rathkopf, M. R. Zika

May 13, 2014

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Estimating the Impact of Advanced Architectures on ASC Multiphysics Codes in the Absence of an Exascale Initiative

C. H. Still, C. J. Clouse, M. G. McCoy, J. R. Neely, B. S. Pudliner,
J. A. Rathkopf, and M. R. Zika
Lawrence Livermore National Laboratory

The Eras of High Performance Computing and Lessons Learned

Figure 1 depicts the three eras in High Performance Computing as experience by the National Nuclear Security Administration and its predecessors thus far. Each era brought lessons that are largely relevant today as we consider future architecture technology. We review them briefly here for reference in the following sections.

Large sequential processing machines populated the first era, dubbed Mainframe Computing. One instruction was executed at a time in serial fashion when a code ran. In this era, memory capacity (how much data could be stored) was sometimes

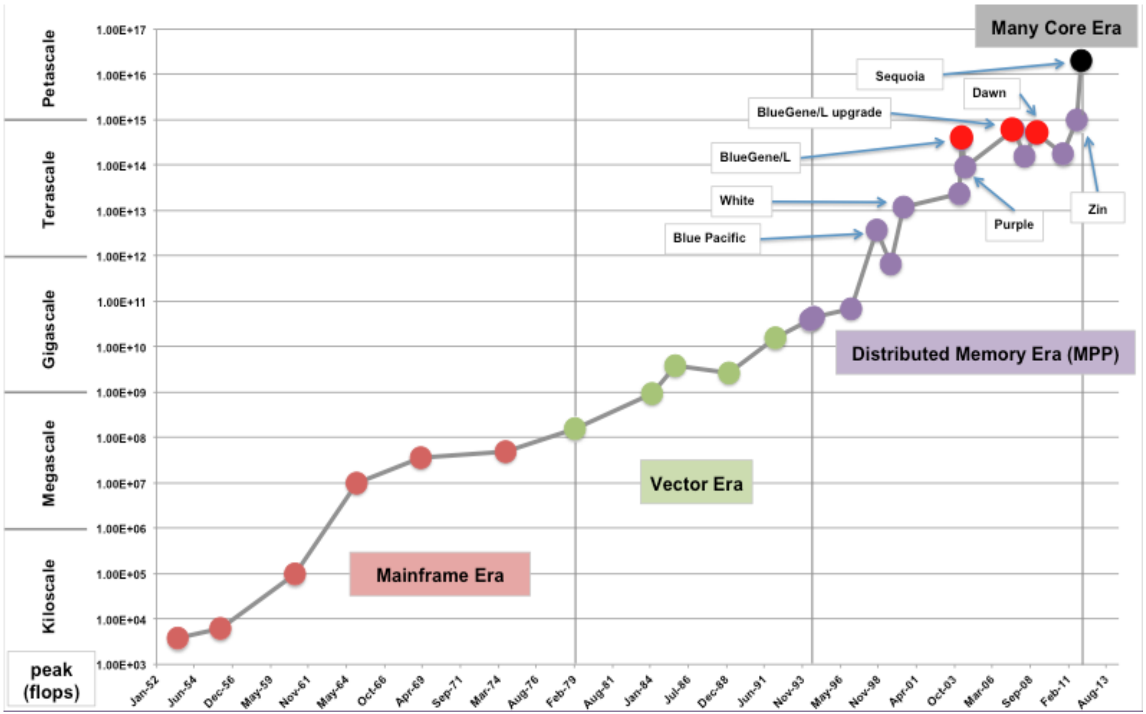

Figure 1. Eras of High Performance Computing

an issue, leading to out-of-core methods which kept part of the data on secondary storage, and relied on exchanging memory resident data that was not immediately needed for the non-resident data required to continue computing. As computer memories grew, larger calculations could fit entirely in the resident memory, and out-of-core calculations were reserved for the largest simulations needed.

Towards the end of the CDC 7600, vector computing began to emerge as the direction for future hardware. The first major design was the CDC Star-100, which offered a large performance boost for vector operations, but very little (if any) performance improvement for scalar operations. Moving to vector processing required modifying algorithms to process more than one data item at a time. However, some operations are inherently scalar calculations and could not be vectorized. In the multiphysics codes of that time, approximately 30% of the calculation could be vectorized, thus the overall simulation performance was limited by the time to complete the scalar portions (Amdahl's Law), leading to very little overall improvement. The Cray-1 was designed differently. Armed with the knowledge that scalar processing performance was a concern, that machine was designed for a much better balance: much improved scalar performance, even though the vector performance was not as high as the Star. The overall result was much improved simulation runtimes.

As the gains available in vector performance began to wane, large expensive vector processors were replaced by increased numbers of cost-effective serial processors, connected by a fast network. Each processor had a local memory (hence Distributed Memory Computing), so that physics problems had to be partitioned into domains, which would then be mapped onto separate processors. Any non-resident data needed for a computation would have to be explicitly communicated from another processor. Poor partitioning of the problem would lead to an imbalance in the work assigned to each processor, leaving some processors with too little work, and others with too much. The standardized Message Passing Interface (MPI) was designed to be a portable way to implement the explicit messages. Algorithms had to be redesigned again, undoing the vectorization of the previous era, and employing efficient communicating serial processes instead. In particular, optimizations done for higher vector performance were often contrary to optimizations needed for cache efficiency; without algorithm redesign, performance would have poor on the new processors. As with the out-of-core issues before, performance was limited by the amount of data sent, the frequency of the messages, and the distances of the message targeted processors. The advantage was that data motion could be controlled explicitly.

By the end of the Distributed Memory era, memory available on each node had become quite large, and the bandwidth available was more than sufficient to keep the processor supplied with the data needed for multiphysics calculations. Furthermore, the processors had become much more advanced. A small amount of memory was located inside the processor (a cache) so that data could be reused (avoiding extra store and fetch operations), avoiding extra memory motion. In addition, more recent server chips could handle memory latency (time delay between when data is requested by the processor, and when it begins to arrive) implicitly via out-of-order execution: when the current instruction stalls awaiting data, the processor is able to examine and execute downstream instructions if the necessary data is already resident. The gold standard for these systems was the IBM Power5-based ASC Purple machine (in service, 2005-2010). The system was extremely well-balanced for computation, but not very energy efficient. If the out-of-order execution was insufficient to manage memory latency, threads (light-weight processes between which processing could quickly shift) could be employed. The machine was so well-provisioned that threading on ASC Purple was never found to improve performance.

With the advent of the Blue Gene line, light-weight processors were introduced with the goal being higher energy efficiency. Unlike the server chips used in other distributed memory machines, the BG cores executed instructions in-order, and thus threading was required to manage memory latency, and SIMD/vectorization was introduced. In the last of the Blue Gene line (BG/Q), each node has 16 processors, each with 4 hardware threads and 4-wide SIMD, sharing a memory, and connected to the other nodes by a fast network. The cores are capable of processing faster than data can be moved into the cores. Thus, memory bandwidth (the rate at which data can be moved into a processor) is becoming the performance-limiting constraint. This sets the stage for the emerging next era of advanced architectures.

Emerging Architectures

In addition to the manycore model (described by the Blue Gene series above), Figure 2 depicts well-known architectures and several other emerging advanced

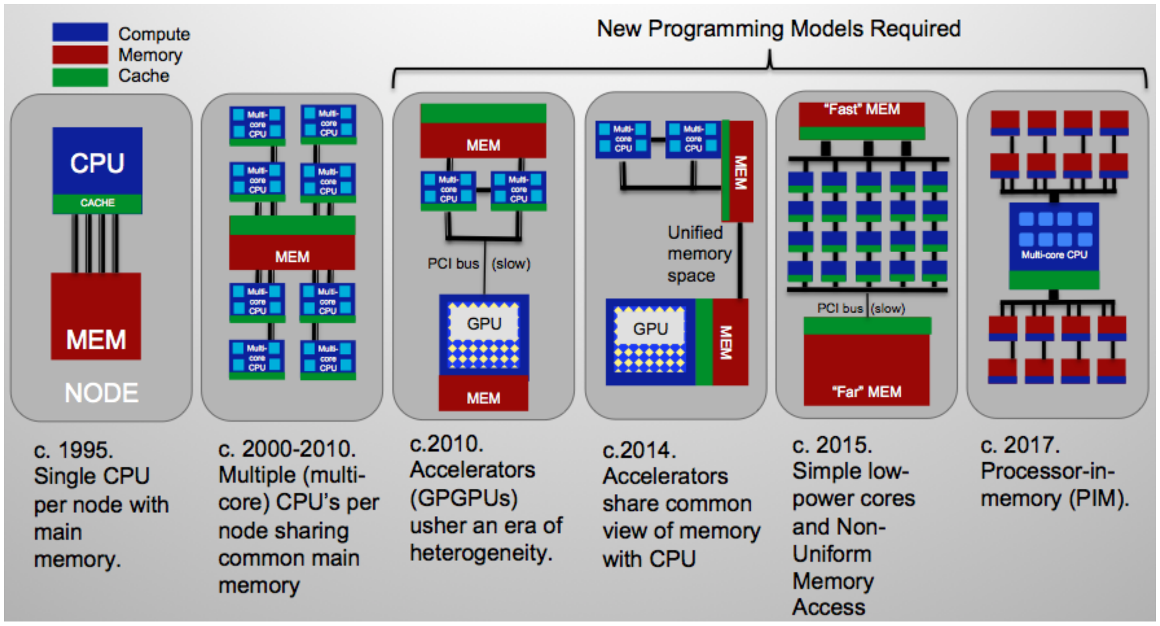
Compute
Memory
Cache
New Programming Models Required
CPU
CACHE
MEM
NODE
Multi-core CPU
MEM
PCI bus (slow)
GPU
Unified memory space
"Fast" MEM
"Far" MEM
Multi-core CPU
c. 1995. Single CPU per node with main memory.
c. 2000-2010. Multiple (multi-core) CPU's per node sharing common main memory
c.2010. Accelerators (GPGPUs) usher an era of heterogeneity.
c.2014. Accelerators share common view of memory with CPU
c. 2015. Simple low-power cores and Non-Uniform Memory Access
c. 2017. Processor-in-memory (PIM).

Figure 2 The Evolution of HPC Node Architectures

architectures. The first two blocks in the figure describe node architectures prevalent in the Distributed Memory computing era. Each of the architectures listed under the grouping "New Programming Models Required" will require changing the current algorithms (for distributed memory architectures) in order to execute on the new architecture. In particular, the explicit-communication, serial processing paradigm that has served so well through the Distributed Memory Computing era (ie, the "ASCI era") is not sufficient to address the new architectural characteristics.

The new characteristics not only describe how and where computation is done, but also where data is stored, and how it is accessed.

In a GPU (block 3), the streaming multiprocessors work in a symmetric multithreading (SIMT, or single-instruction, multiple-thread) fashion. Fast memory is located on the GPU accelerator directly, but capacity is limited, so the bulk of the calculation is stored off the accelerator. Access to the accelerator from the bulk memory is via a slow link, compared to the on-package memory (at 10x higher bandwidth). Thus, reminiscent of out-of-core algorithms, performance is dramatically affected by data placement and the ability to overlap data transfers with computation. In newer GPUs (block 4, e.g., nVidia CUDA 6 supports this with the Maxwell accelerator), there is a unified virtual address space so that data transfers can be handled implicitly, rather than explicitly, assuming a compiler (perhaps with directives inserted into the source-code by the programmer) can understand the data dependencies sufficiently to optimize for data placement. Since the GPU shares one instruction-stream, optimizing data motion to coalesce memory requests gives the best performance.

The manycore architecture (block 5) is reminiscent of the Blue Gene architecture, except that memory is split into two classes: a smaller "fast" memory with high bandwidth located on the manycore package directly, and a much larger "far" memory accessed via a slow link. This two-level memory is analogous to that seen on the GPU architectures. Unlike the GPU, each of the cores has its own instruction stream, and is fetching its own data. Thus, coalescing data reads is not necessary, but managing resource contention is required.

The final emerging architecture is still a research topic: processing in memory. This is perhaps the most energy efficient manner for processing, but requires a complete algorithm redesign: instead of bringing the data into the compute engine, the instruction stream is shipped off into the memory processor. High performance is obtained by distributing the data in such a way as to keep reusing the same instruction stream as long as possible. Calculations that cannot be performed directly in memory would be done in a centralized processor in a more traditional manner (data fetched from memory, operations performed, results stored back into memory).

Effect of Advanced Architectures on Multiphysics Performance

Comparing the future architectures described above, a few common characteristics emerge, in particular, memory locality, memory capacity and memory bandwidth. These characteristics are becoming the dominant constraints for overall code performance. In particular, the memory capacity determines how many domains can be loaded onto a node, and thus how many MPI tasks can be used. Within an MPI task, the memory bandwidth determines how many floating point operations can be supported by the availability of data.

Figure 3 shows projections for code performance on advanced architectures for the rest of the decade. In this analysis, we are assuming that the computing platform is constrained by a $180M procurement cost. Using responses to the E7 Request for

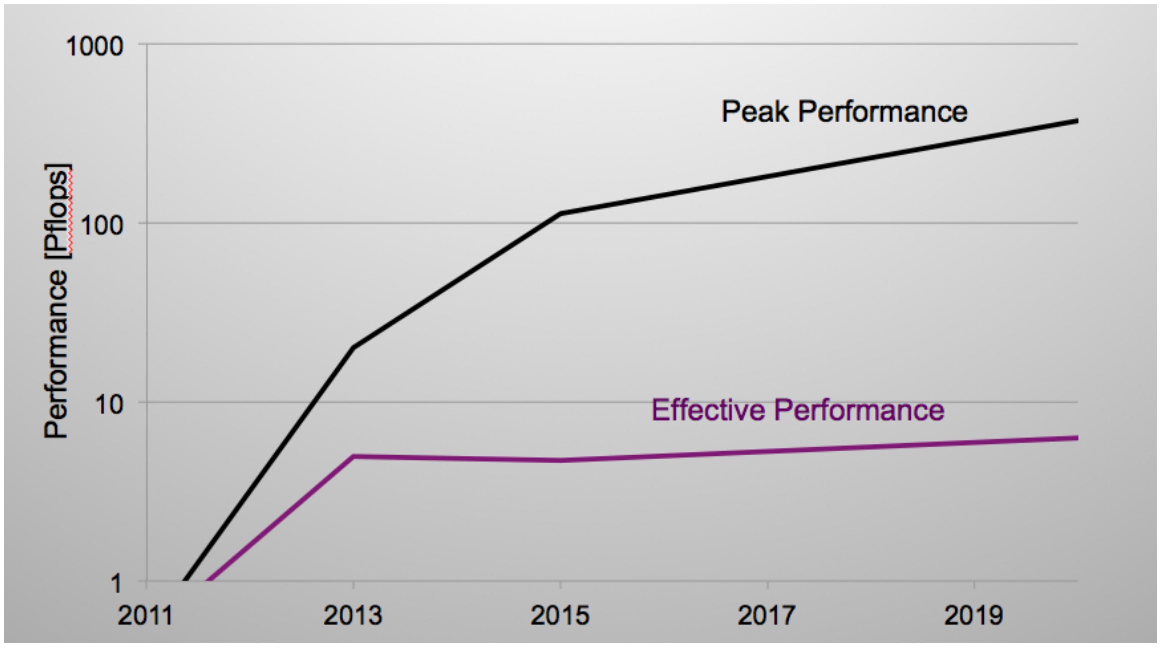

Figure 4 Projected Effective Performance of Multiphysics Codes on Future Architectures

Information on Exascale Architectures (2011), we extracted the memory capacity and bandwidth data available on such a platform in 2015-2020. We further assumed that the Exascale Initiative was *not* funded, and that the only investments were those available within the baseline funding of the ASC Program. Thus, there is little leverage on vendor roadmaps and technology beyond what would be available in the typical Non-Recurring Engineering (NRE) portion of the platform procurement. The vendors candidly responded to the RFI including the scenario without national investment. The historical trend and future projection of the B:F (ratio of memory bandwidth to floating point rate, ie the ratio of data transfer rate to data processing rate) and memory capacity are shown in Figure 4. Finally, we assumed that the only modifications to the ASC multiphysics codes would be those capable of being carried out in the course of the baseline program (ie, no additional investments to ASC for the Integrated Codes). Finally, performance is normalized to ASC Purple equivalence, ie. ASC multiphysics code performance on the 100 TFlop ASC Purple machine.

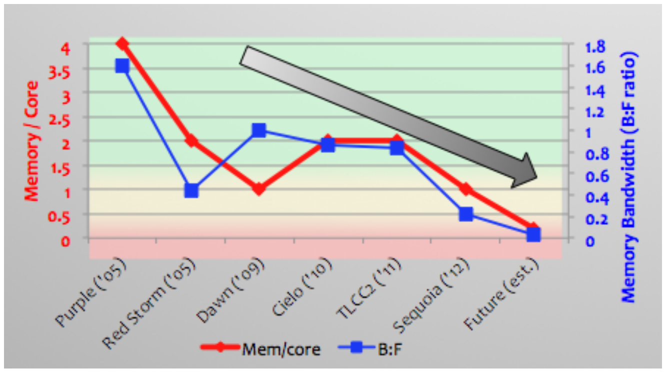

Figure 3 Memory Capacity and B:F (memory bandwidth to floating point rate) for ASC Platforms

The 2011 and 2013 data is taken to be best possible code performance based on available data, as of June 2013, on ASC Cielo and ASC Sequoia. In particular, on ASC Sequoia, this assumes vectorization up to the memory bandwidth limit. On future machines, the floating-point capability is growing far faster than the memory bandwidth to support the computation, leading to the ever-widening gap between peak performance and effective performance. The amount of threading (increasing from 2-16), data locality (working set size reduces from 2GB per domain to 512MB per domain), and amount of data shared (from 0 to 500 MB) within an MPI task is increasing over time, representing the expectation of improved code optimizations, which does lead to a slight increase in effective performance through the end of the decade. However, the overall trend is essentially flat, with less than a factor of 2 in performance improvement from 2013 to 2020.

Final Thoughts

Without significant investment into both the hardware vendors and the ASC Program, closing the performance gap will not be possible. Even with investment, the challenges facing the industry are well documented. Solutions which trade (for example) improved memory bandwidth for increased memory latency introduce differing constraints on code performance. Thus, a close coupling between the vendors and the multiphysics code developers (through a co-design process) is a critical component to ensure not only that solutions in the codes are possible, but that practical implementations are developed to meet the constraints imposed by the vendors' design trade-offs.

Future versions of this document will incorporate updated information on vendor technology.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.